QuickRanking: Fast Algorithm For Sorting And Ranking Data

Laurent Laurent Ott Ott

QuickRanking is an algorithm that allows in a same processing, to sort the data and to return their ranking, that is to say their rank in the sorting.

For faster processing, QuickRanking does not move data, but stores for each data, where the following data is. This method is explained in annex 1.
The source code written in VBA is in annex 2.

To evaluate the effectiveness of QuickRanking, I have reproduced at annex 3 the source code for QuickSort, and in annex 4 an "enriched" version of QuickSort, called QuickSort_AndRank, which like QuickRanking, does not move the data directly but moves the references of the indices of the data. Hence a gain of time on the processing of the voluminous alphanumeric data because we move integers of 4 bytes and no more strings of character of varying lengths. This requires additional memory capacities, but therefore allows to use these reference tables to return an order of classification in addition to the sorting, just as QuickRanking does.

I have compared these three algorithms - QuickSort, QuickSort_AndRank, and QuickRanking - on lists of random data; the results are synthetised in a table in annex 5. My analysis is the following:

If the classification order is not necessary:
- Sorting of digital data: QuickSort is the fastest algorithm except when the data at the beginning of the list are already classified, or if the minimum or maximum value is often represented. In these cases, QuickRanking takes the advantage.
- Sorting of alphanumeric data: the more the number of characters of the data to be sorted is important, the more the algorithms work by reference, QuickSort_AndRank and QuickRanking, are effective. QuickSort must only be used if the memory capacity does not allow to use the other two algorithms which require more resources.

If the classification order is necessary:
- Sorting and classification of digital data: QuickSort_AndRank is more efficient than QuickRanking only on large lists of data. QuickRanking takes the advantage in other cases. Advantage emphasised when the data at the beginning of the list are already classified, when the list is made up of few different values, or when the minimum or maximum value is often represented.
- Sorting and classification of alphanumeric data: QuickRanking is faster than QuickSort_AndRank. Advantage emphasised on the processing of large chains of characters.

Conclusion: ideally random list or highly classified list, large data or reduced size data, numeric or alphanumeric data, the reality is often between these extremes. In practice, QuickRanking may provide an interesting alternative to QuickSort...

Remarks:
- QuickRanking offers optional complementary analyzes to accelerate the processings on the lists where the data have many equalities, or when data are following. This option has not been used in the tests because they are based on random data. Annex 6 presents a function that determines, after analysis of a sample of data from the list, whether or not to enable this option.
- QuickRanking also allows you to reverse only the order of classification, without sorting all the data, which reduces the processing time. This method is useful for instance when you are looking for the first 10 elements of a list, or the nth item. However, the equalities are not managed in this method.
- The attached Excel file takes up these algorithms.
- In the tests presented, QuickSort and QuickSort_AndRank use a fixed pivot.
- The tests were done on a PC office automation in VBA in EXCEL. The results are given for information; the objective is to bring out a trend. These results need to be confirmed in other programming languages and with other operating systems. I am looking for volunteers, if you are tempted by the adventure: Laurent.ott8@orange.fr
- Thanks to Christiane Bonin and Patrick for this translation.

Let's sort the 12 data in the table below to understand this sorting method, which to save processing time, does not move data, but stores for each date where the following data is:

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90

Let us analyze the first two elements: Here we know that the item 0, which is worth 3, is the minimum value, and the item 1, which is worth 9, is the maximum value.
Therefore the item following the item 0 is the item 1. We can store this information:

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	1	x										

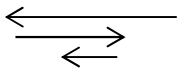
To simplify the reading of the table, the minimum value is represented in green and the maximum value is in orange.

To classify the item 2, which is worth 5: We read the minimum value, the item 0. The value 3 is less than 5, therefore we read the next item, the item 1, which is worth 9. Too large this time. Therefore 5 becomes the value that will follow 3 and 9 will be the value that follows 5.
Which updates our table of "next item":

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	2	x	1									

Let's classify the item 3 which is worth 2, lower value to our former minimum value which was the item 0 of value 3. Therefore an exchange is made. And the item 0 will be the value which will follow the item 3.
No change for the other data already analysed:

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	2	x	1	0								



These arrows indicate the reading order of the table starting from the minimum reference, and in reading the "next item".

Let us go to item 4 which is worth 15, value superior to our old maximum value, the item 1 which is worth 9. Therefore an exchange is made. The item 4 will be the value which will follow the item 1.
No change for the other data already analysed:

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	2	4	1	0	x							

The item 5 is worth 7, superior value to our minimum value of reference item 3, which is worth 2. Therefore we read the next item, the item 0, which is worth 3. Always less than 7, so we read the next item: item 2, which is worth 5. We continue with the next item, 1, which is worth 9. This time we can break out of the loop and update the "Next item": the item following the figure 7 is therefore the item 1, and the new "next item" to the item 2 is no more 1 but 5:

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	2	4	5	0	x	1						

Same principles to classify the item 6 which is worth 8: After reading the values from the table starting from the minimum value item and using the "next item", we find that it is located between the item 5 which is worth 7 and the item 1 which is worth 9. Therefore its "next item" is the item 1, and "next item" of the item 5 is modified: 1 is replaced by 6.

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	2	4	5	0	x	6	1					

Note: here a shortcut is possible to quickly sort the item 6. Actually 8 is superior or equal to the last value analysed, which was worth 7, and is inferior or equal to the value of the next item of the last value analysed, which is worth 9. Therefore the next item of the item 6 is the next item of the last value analysed. And the next item of the last value analysed becomes the item 6.
This test is very efficient on the partially sorted lists or if there are many equalities, but is counter-productive on random lists. That is why this test is optional in QuickRanking in order not to run it on lists of data that you know being random, and thus save the processing time.
Annex 6 presents a function that determines, after analysis of a sample of data from the list, whether or not to enable this option.

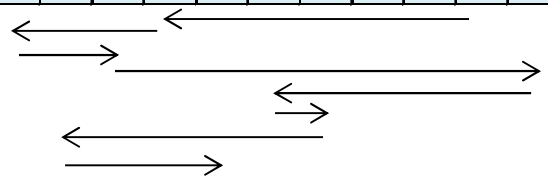
Continue the classification and if you get this table, is that you have understood:

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	2	4	10	0	8	6	1	11	7	3	5	x

To read the data in ascending order, it must be from the minimum reference, the item 9, the 0, and read the "next item ": item 3, which is worth 2, the item 0 which is worth 3, the item 2 which is worth 5, and so on...

The data is sorted without any movement is made.
Only problem: to know the classification of an element, it sometimes takes many readings.
For example, to classify the item 12 which is worth 10, here is the path followed, either 8 readings, to find that the item 1 is the last inferior item:

Item :	0	1	2	3	4	5	6	7	8	9	10	11	12
Value :	3	9	5	2	15	7	8	20	17	0	5	90	10
Next item :	2	4	10	0	8	6	1	11	7	3	5	x	

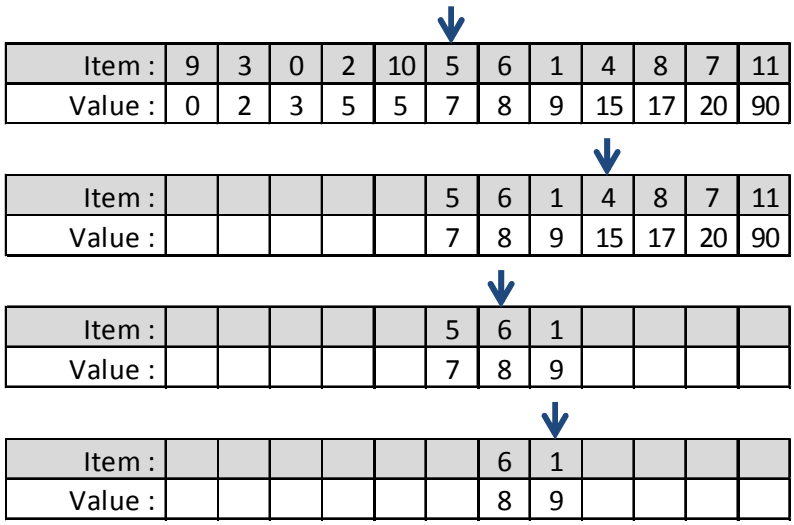


You guess that with a table of several hundreds of elements, certain values to classify will require a number of impressive reading ... which soars the processing time.

To save time, an already classified data table must be kept:

Item :	9	3	0	2	10	5	6	1	4	8	7	11
Value :	0	2	3	5	5	7	8	9	15	17	20	90

Which allows you to make a dichotomous search to find the closest value of the element that we want to classify, which is worth 10, in only 4 readings:



Item :	9	3	0	2	10	5	6	1	4	8	7	11
Value :	0	2	3	5	5	7	8	9	15	17	20	90

Item :						5	6	1	4	8	7	11
Value :						7	8	9	15	17	20	90

Item :						5	6	1				
Value :						7	8	9				

Item :							6	1				
Value :							8	9				

This method takes all its power on a very large table. For example, on a table of 10,000 elements, 14 readings only allow you to find the searched item, or approximately Log (*n*) / Log (2), where n is the number of elements in the table. A comparison with the hundreds of required readings by reading one by one "next item".

The generation of this data table already classified being time consuming, it will be carried out from time to time. But even incomplete, this table allows you to move rapidly to the nearest item and then go on to a regular processing by reading the "next item" one by one, until you find the correct item.
Consider this table as a shortcut to quickly arrive close to the final destination, and not necessarily to the final destination.

We must arrive at a good compromise so that the time spent to generate this table, is profitable by the time gained by dichotomy research.
I have adopted the principle to update the table when the number of readings of the "next item" is equivalent to the number of processed data.

A mathematical analysis should allow to find the formula for an optimal generation.
I am open to your proposals.

Very Important Note : The search algorithm of dichotomous QuickRanking is different from conventional algorithms based on a loop of type " Do ... Loop While beginning ≤ end ", including a condition of equality to break out prematurely of the loop when the value sought is found.
To speed up processing QuickRanking proceeds in two steps:

- The first step is a loop " For… next " made of a number of times inferior to the number calculated by the formula Log(*n*) / Log(2) which gives the maximum number of loop that will have to be done to find the solution in a dichotomous search. This is sufficient to approach the solution.
- The second step is a loop that runs from the last inferior solution and which, this time, researches the exact solution.

This approach allows you to limit the number of conditional tests, which are time-consuming processing, and, most important, allows QuickRanking to be faster than if it used a conventional dichotomous research.

QuickRanking sorts the data, but also returns a rank order.

Let's go back to the table already seen:

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	2	4	10	0	8	6	1	11	7	3	5	x

The reading of the next items gives the sorting of the elements: 0, 2, 3, 5, 5, 7, 8, 9, 15, 17, 20, 90.
This reading also allows to obtain the rank order of the elements:

The first element is the item 9. In an annexed table that serves as a ranking order, we put 1 for the item 9.

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	2	4	10	0	8	6	1	11	7	3	5	x
Rank :										1		

The following item is item 3, we put 2 in the rank order of the item 3.

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	2	4	10	0	8	6	1	11	7	3	5	x
Rank :				2						1		

The following item is item 0, we put 3 in the rank order of the item 0.
And so on to obtain this table:

Item :	0	1	2	3	4	5	6	7	8	9	10	11
Value :	3	9	5	2	15	7	8	20	17	0	5	90
Next item :	2	4	10	0	8	6	1	11	7	3	5	x
Rank :	3	8	4	2	9	6	7	11	10	1	5	12

The item 0 is worth 3, it is the 3rd element of the sorted list.
The item 1 is worth 9 it is the 8th element of the sorted list.
The item 2 is worth 5 it is the 4th element of the ordered list...

Inversely, you can know the value of the nth element of a list: List(rank(n))
List(rank(9)) = 15. The 9th item in the list is worth 15.

The rank order is convenient to fill a table of this type:

Participants	Examination 1	Rank	Examination 2	Rank	Total	Rank
Mathieu	15,24	3	12,65	4	27,89	3
Sylvie	14,15	4	17,45	1	31,60	2
Nathalie	16,24	2	8,59	5	24,83	5
Edouard	17,50	1	14,15	2	31,65	1
Luc	13,37	5	13,80	3	27,17	4

But the exercise may be more complicated in some cases, as in the table below:

Item :	0	1	2	3	4	5	6	7	8
Value :	8	8	9	8	10	8	8	10	5
Next item :	1	2	4	0	7	3	5	x	6
Rank :	5	6	7	4	8	3	2	9	1

The elements of value 8, item 0, 1, 3, 5 and 6, have an inconsistent ranking order. Nothing can justify that the item 6 is ranked 2nd whereas the item 5 is ranked 3th.

In practice, either we attribute the same ranking to all equal values (Method 1):

Item :	0	1	2	3	4	5	6	7	8
Value :	8	8	9	8	10	8	8	10	5
Next item :	1	2	4	0	7	3	5	x	6
Rank :	2	2	7	2	8	2	2	8	1

Here there is a first, and 5 second ex-aequo. The element of value 9 (item 2) is 7$^{th.}$

Either we keep the original order to classify the equal values (Method 2):

Item :	0	1	2	3	4	5	6	7	8
Value :	8	8	9	8	10	8	8	10	5
Next item :	1	2	4	0	7	3	5	x	6
Rank :	2	3	7	4	8	5	6	9	1

Here each element has a separate row. There is no ex-aequo even for the elements of the same value. In case of equality, it is the element at inferior item that is prioritised.

QuickRanking can manage both ranking methods, as well as the ascending or descending order.

The function accepts four arguments, in addition to sort the table:

- *OrdreCroissant (Ascending order)* :
  - If it is True, the sorting is done by ascending order.
  - If it is False, the sorting is descending.
- *ModeClassement (ranking mode)* indicates the ranking mode to return:
  - 0 for no ranking. The data are sorted only.
  - 1 to classify the data by applying the same rank to equal data.
  - 2 to classify the data by assigning a different rank to equal data, respecting their original position.
  - 3 to return the ranking, without sorting the data or managing any equality.
  - 4 to return the data list without duplicates. The data are sorted but are not ranked
- *NiveauTest, (level test)* determines whether or not to perform additional testing, to accelerate the processings on the lists where the data have many equalities, or when the data follow, see annex 6.
  - If it is worth -1 (or True), additional tests are performed.
  - If it is worth 0 (or False), no additional test is performed.
  - If it is worth between 1 and 100: a sample of the data is analysed by activating additional tests. If these tests prove successful, that is to say that the success rate is higher than in past argument *NiveauTest*, then the option will be enabled (the argument then takes the value -1).
- *TauxTest, (rate test) will contain the percentage of effective additional tests (integer between 0 and 100). This argument is used by the "TesterNiveauQR"* function to test whether or not to activate the option of additional tests. In principle this argument is not used in the regular calls to the function, except perhaps by curiosity.

By default, QuickRanking sorts the data in ascending order and rank them by applying the same rank to equal data (Method 1). The option of additional testing is activated when the efficiency rate is at least 15 %.

```
'-------------------------------------------------------------------------------
Public Function QuickRanking(ByRef TabDonnées() As Variant, _
                Optional ByVal OrdreCroissant As Boolean = True, _
                Optional ByVal ModeClassement As Byte = 1, _
                Optional ByRef NiveauTest As Long = 15, _
                Optional ByRef TauxTest As Long = 0) As Variant
'-------------------------------------------------------------------------------
' TabDonnées : Tri les données passées en argument et modifie TabDonnées.
' OrdreCroissant : Si vaut True alors ordre croissant, sinon ordre décroissant.
' ModeClassement : 0 = Tri, Pas de classement.
'                  1 = Tri + Classement des données, les données égales ont le même ordre.
'                  2 = Tri + Classement des données, l'ordre des données égales
'                      respecte l'ordre d'origine.
'                  3 = Uniquement Classement des données, et sans gestion des égalités.
'                  4 = Tri sans doublon, et sans Classement.
' NiveauTest : False (0) = Pas de test complémentaire,
'              True (-1) = Contrôle les égalités et les suites.
'              >0 et <100 = Lance le test pour savoir s'il faut activer ou non l'option,
'                           où NiveauTest représente le taux de conformité (de 1 à 100)
'                           pour que l'activation de l'option soit considérée utile.
'                           NiveauTest sera alimenté du resultat obtenu (Vrai ou Faux).
' TauxTest : Contiendra le taux (0 à 100) des tests efficaces. Utilisé pour tester l'option.
'-------------------------------------------------------------------------------

' S'il faut lancer le test du choix de l'option pour NiveauTest. NiveauTest contient
' le pourcentage de réussité désiré des tests complémentaires pour activer l'option:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If NiveauTest > 0 Then NiveauTest = TesterNiveauQR(TabDonnées(), NiveauTest)

' Bornes du tableau des données d'origine:
Dim TabDébut As Long, TabFin As Long
On Error Resume Next ' Si aucune donnée à trier.
TabDébut = LBound(TabDonnées)
TabFin = UBound(TabDonnées)

' Initialisation du tableau du classement des données:
ReDim Ref(TabDébut - 2 To TabFin) As Long

' Si rien à trier alors quitte:
If Abs(TabFin - TabDébut) < 1 Then QuickRanking = Ref(): Exit Function

' Initialisation des variables pour le traitement de tri:
Dim Tps As Variant, ValMini As Variant, ValMaxi As Variant
Dim i As Long, n As Long, j As Long, Anc As Long, l As Long
Dim RefMini As Long, RefMaxi As Long, MaxiRac As Long, MiniRac As Long
Dim NbPassage As Long, Début As Long, Fin As Long
Dim NbRechercheDicho As Long, MaxiDoWhile As Long, Compteur As Long

' Initialisation du tableau des données déjà classées:
ReDim TabTps(TabDébut - 2 To TabFin) As Long
MaxiRac = TabDébut
NbPassage = TabFin

' Configure le classement des 2 premiers éléments:
If TabDonnées(TabDébut) > TabDonnées(TabDébut + 1) Then n = 1
RefMini = TabDébut + n
RefMaxi = TabDébut + 1 - n
Ref(TabDébut) = TabDébut + 1
Ref(TabDébut + 1) = RefMaxi
ValMini = TabDonnées(RefMini)
ValMaxi = TabDonnées(RefMaxi)
```

```
' Si l'option des tests complémentaires est à Vrai (-1):
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If NiveauTest = True Then

' Boucle sur les éléments à classer en effectuant les tests complémentaires:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
For n = 2 + TabDébut To TabFin
    Tps = TabDonnées(n)

   ' Controle le débordement du mini:
    Do Until Tps > ValMini           ' Plus rapide que If Tps <= ValMini Then... End If.
        Ref(n) = RefMini             ' La données suivante de n est l'ancien minimum.
        RefMini = n                  ' Mémorise qui est le nouveau minimum.
        TabTps(TabDébut) = n         ' Le 1er élément du tableau de recheche dicho.
        MiniRac = TabDébut           ' Le minimum pour la mise à jour du tableau de recherche dicho.
        Anc = TabDébut               ' Position du dernier élément analysé dans le tableau de recherche d
        ValMini = Tps                ' Nouveau minimum.
        GoTo Element_Suivant         ' Fin du traitement de n.
    Loop

    ' Controle le débordement du maxi:
    Do Until ValMaxi > Tps           ' Plus rapide que If Tps >= ValMaxi Then... End If.
        Ref(RefMaxi) = n             ' La donnée suivante de l'ancien maximum est n.
        Ref(n) = n                   ' La donnée suivante de n est n.
        RefMaxi = n                  ' Mémorise qui est le nouveau maximum.
        MaxiRac = MaxiRac + 1        ' Dernière positon dans le tableau de recherche dicho.
        TabTps(MaxiRac) = n          ' Le tableau de recherche dicho peut être alimenté.
        Anc = MaxiRac                ' Position du dernier élément analysé dans le tableau de recherche dic
        ValMaxi = Tps                ' Nouveau maximum.
        GoTo Element_Suivant         ' Fin du traitement de n.
    Loop

    ' Mise à jour du tableau des données déjà classées:
    While NbPassage > n                          ' While est plus rapide que If... Then... End If.
        i = TabTps(MiniRac)                      ' Boucle depuis la position du plus petit élément analysé,
        If MiniRac = TabDébut Then i = RefMini ' ou boucle depuis la position du minimum.
        For j = MiniRac To n
            TabTps(j) = i                        ' Mémorise la position de l'élément.
            i = Ref(i)                           ' Position de l'élément suivant.
        Next j
        MaxiRac = n - 1                          ' Le dernier élément n'est pas utilisé.
        MiniRac = MaxiRac                        ' Efface la position du plus petit élément analysé.
        NbPassage = n * 0.3                      ' Initialise le nombre de passages pour mise à jour du tabl
        NbRechercheDicho = Log(n) / Log(2)       ' Nombre maximum de recherches possibles dans le tableau di
        If NbRechercheDicho > 5 Then MaxiDoWhile = NbRechercheDicho ' Limite pour les suites non contigües
        Début = TabDébut: Fin = MaxiRac          ' Bornes pour la recherche Dichotomique.
        GoTo RechercheDichotomique
    Wend

    ' Bornes pour la Recherche Dichotomique dans le tableau des données déjà classées:
    Début = TabDébut: Fin = MaxiRac

    ' Tests complémentaires (égalités et suites immédiates):
    Do Until TabDonnées(n - 1) > Tps             ' Si n est >= dernier élément analysé.
        Début = Anc                              ' Borne de début pour la recherche dicho.
        Do Until Tps > TabDonnées(Ref(n - 1))    ' Si n est <= élément suivant du dernier élément analysé.
            Ref(n) = Ref(n - 1)                  ' Echange de la donnée suivante de n et de l'ancien élémer
            Ref(n - 1) = n                       ' n devient la donnée suivante de l'ancien élément.
            TauxTest = TauxTest + 1              ' Nombre de tests efficaces.
            GoTo Element_Suivant                 ' Fin du traitement de n.
        Loop
        GoTo RechercheDichotomique               ' Passe à la recherche avec la nouvelle borne de début.
    Loop
    Fin = Anc                                    ' Borne de fin pour la recherche dicho.
```

```
    ' Recherche Dichotomique dans le tableau des données déjà classées:
RechercheDichotomique:

    For j = 4 To NbRechercheDicho  ' Plus rapide que Do...Loop While Début + 2 < Fin
        i = (Début + Fin) / 2      ' Calcule le milieu.
        If Tps > TabDonnées(TabTps(i)) Then Début = i Else Fin = i
    Next j
    While TabDonnées(TabTps(Début + 1)) < Tps: Début = Début + 1: Wend

    Anc = Début     ' Solution.
    i = TabTps(Anc) ' Plus proche donnée inférieure connue.
    While Anc < MiniRac: MiniRac = Anc: Wend ' Plus rapide que If Anc < MiniRac Then MiniRac = Anc

    ' Boucle sur les indices suivants pour trouver le classement du nouvel élément:
    Compteur = 0
    Do
        j = i                       ' Dernière Solution.
        i = Ref(i)                  ' Indice suivant
        Compteur = Compteur + 1     ' Compte le nombre de passages infructeux.
    Loop While Tps > TabDonnées(i)  ' Sort si la valeur de l'indice suivant >= Tps.
    NbPassage = NbPassage + Compteur

    Ref(n) = Ref(j)        ' Qui est la donnée suivante de n.
    Ref(j) = n             ' n devient la donnée suivante de l'ancien élément.

    ' Gestion des suites non contigües:
    While Compteur > MaxiDoWhile
        TabTps(Anc - 2) = TabTps(Anc - 1)
        TabTps(Anc - 1) = TabTps(Anc)
        TabTps(Anc) = n
        TabTps(TabDébut) = RefMini
        Compteur = MaxiDoWhile
    Wend

Element_Suivant:
Next n

' Alimente le taux d'efficacité des tests complémentaires:
TauxTest = TauxTest * 100 / (TabFin - TabDébut)
```

```
' Si l'option des tests complémentaires est à Faux (0):
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Else

' Boucle sur les éléments à classer sans effectuer les tests complémentaires:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
For n = 2 + TabDébut To TabFin
    Tps = TabDonnées(n)

   ' Controle le débordement du mini:
    Do Until Tps > ValMini           ' Plus rapide que If Tps <= ValMini Then... End If.
        Ref(n) = RefMini             ' La données suivante de n est l'ancien minimum.
        RefMini = n                  ' Mémorise qui est le nouveau minimum.
        TabTps(TabDébut) = n         ' Le 1er élément du tableau de recheche dicho.
        MiniRac = TabDébut           ' Le minimum pour la mise à jour du tableau de recherche dicho.
        Anc = TabDébut               ' Position du dernier élément analysé dans le tableau de recherch
        ValMini = Tps                ' Nouveau minimum.
        GoTo ST_Element_Suivant        ' Fin du traitement de n.
    Loop

    ' Controle le débordement du maxi:
    Do Until ValMaxi > Tps           ' Plus rapide que If Tps >= ValMaxi Then... End If.
        Ref(RefMaxi) = n             ' La donnée suivante de l'ancien maximum est n.
        Ref(n) = n                   ' La donnée suivante de n est n.
        RefMaxi = n                  ' Mémorise qui est le nouveau maximum.
        MaxiRac = MaxiRac + 1        ' Dernière positon dans le tableau de recherche dicho.
        TabTps(MaxiRac) = n          ' Le tableau de recherche dicho peut être alimenté.
        Anc = MaxiRac                ' Position du dernier élément analysé dans le tableau de recherche
        ValMaxi = Tps                ' Nouveau maximum.
        GoTo ST_Element_Suivant        ' Fin du traitement de n.
    Loop

    ' Mise à jour du tableau des données déjà classées:
    While NbPassage > n                          ' While est plus rapide que If... Then... End If.
        i = TabTps(MiniRac)                      ' Boucle depuis la position du plus petit élément analy
        If MiniRac = TabDébut Then i = RefMini ' ou boucle depuis la position du minimum.
        For j = MiniRac To n
            TabTps(j) = i                        ' Mémorise la position de l'élément.
            i = Ref(i)                           ' Position de l'élément suivant.
        Next j
        MaxiRac = n - 1                          ' Le dernier élément n'est pas utilisé.
        MiniRac = MaxiRac                        ' Efface la position du plus petit élément analysé.
        NbPassage = 0                            ' Initialise le nombre de passages pour mise à jour du
        NbRechercheDicho = Log(n) / Log(2)       ' Nombre maximum de recherches possibles dans le tablea
    Wend

    ' Recherche Dichotomique dans le tableau des données déjà classées:
    Début = TabDébut: Fin = MaxiRac
    For j = 2 To NbRechercheDicho ' Plus rapide que Do...Loop While Début + 2 < Fin
        i = (Début + Fin) / 2       ' Calcule le milieu.
        If Tps > TabDonnées(TabTps(i)) Then Début = i Else Fin = i
    Next j

    Anc = Début      ' Solution.
    i = TabTps(Anc) ' Plus proche donnée inférieure connue.
    While Anc < MiniRac: MiniRac = Anc: Wend ' Plus rapide que If Anc < MiniRac Then MiniRac = Anc

    ' Boucle sur les indices suivants pour trouver le classement du nouvel élément:
    Do
        j = i                        ' Dernière Solution.
        i = Ref(i)                   ' Indice suivant
        NbPassage = NbPassage + 1    ' Compte le nombre de passages infructeux.
    Loop While Tps > TabDonnées(i)   ' Sort si la valeur de l'indice suivant >= Tps.

    Ref(n) = Ref(j)        ' Qui est la donnée suivante de n.
    Ref(j) = n             ' n devient la donnée suivante de l'ancien élément.

ST_Element_Suivant:
Next n

End If
```

```
' S'il faut retourner le Classement sans le tri:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If ModeClassement = 3 Then
    i = TabTps(MiniRac): If MiniRac = TabDébut Then i = RefMini
    For n = MiniRac To TabFin
        TabTps(n) = i
        i = Ref(i)
    Next n
    QuickRanking = TabTps()
    Exit Function
End If

' Fait une copie temporaire du tableau d'origine:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Erase TabTps: ReDim Mémo(TabDébut To TabFin) As Variant
For n = TabDébut To TabFin
    Mémo(n) = TabDonnées(n)
Next n

' Initialisation du tableau du classement si demandé:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If ModeClassement > 0 Then
    ReDim Pos(TabDébut To TabFin) As Long
    ReDim Egalités(TabDébut To TabFin) As Long
End If
' Classe les données dans l'ordre croissant:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If OrdreCroissant = True Then
    i = RefMini
    For n = TabDébut To TabFin
        TabDonnées(n) = Mémo(i)
        i = Ref(i)
    Next n

    ' S'il faut retourner le Classement où les égalités ont le même classement:
    If ModeClassement = 1 Then
        i = RefMini: Anc = i: NbPassage = 1
        For n = TabDébut To TabFin
            Pos(i) = NbPassage: NbPassage = NbPassage + 1
            If Mémo(i) = Mémo(Anc) Then Pos(i) = Pos(Anc)
            Anc = i: i = Ref(i)
        Next n
        QuickRanking = Pos(): Exit Function
    End If

    ' S'il faut retourner le Classement où les égalités distinguent l'ordre d'origine:
    If ModeClassement = 2 Then
        i = RefMini: Anc = i: j = TabDébut: NbPassage = 1
        For n = TabDébut To TabFin
            Egalités(j) = i: Anc = i: i = Ref(i): j = j + 1
            If Mémo(i) > Mémo(Anc) Then
                If j > TabDébut + 1 Then Call QuickSort(Egalités(), TabDébut, j - 1)
                For l = TabDébut To j - 1
                    Pos(Egalités(l)) = NbPassage: NbPassage = NbPassage + 1
                Next l
                j = TabDébut
            End If
        Next n
        If j > TabDébut + 1 Then Call QuickSort(Egalités(), TabDébut, j - 1)
        For l = TabDébut To j - 1
            Pos(Egalités(l)) = NbPassage: NbPassage = NbPassage + 1
        Next l
        QuickRanking = Pos(): Exit Function
    End If
```

```
    ' S'il faut retourner le tri sans doublons (et sans classement):
    If ModeClassement = 4 Then
    NbPassage = TabDébut
    For n = TabDébut + 1 To TabFin
        If TabDonnées(n) <> TabDonnées(n - 1) Then NbPassage = NbPassage + 1
        TabDonnées(NbPassage) = TabDonnées(n)
    Next n
    ReDim Preserve TabDonnées(TabDébut To NbPassage)
        QuickRanking = Pos(): Exit Function
    End If

    QuickRanking = Pos()
    Exit Function

End If

' Classe les données dans l'ordre Décroissant:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
i = RefMini
For n = TabFin To TabDébut Step -1
    TabDonnées(n) = Mémo(i)
    i = Ref(i)
Next n

' S'il faut retourner le Classement où les égalités ont le même classement:
If ModeClassement = 1 Then
    i = RefMini: Anc = i: NbPassage = TabFin - TabDébut + 1
    For n = TabFin To TabDébut Step -1
        Pos(i) = NbPassage: NbPassage = NbPassage - 1
        If Mémo(i) = Mémo(Anc) Then Pos(i) = Pos(Anc)
        Anc = i
        i = Ref(i)
    Next n
    QuickRanking = Pos(): Exit Function
End If

' S'il faut retourner le Classement où les égalités distinguent l'ordre d'origine:
If ModeClassement = 2 Then
    i = RefMini: Anc = i: j = TabDébut: NbPassage = TabFin - TabDébut + 1
    For n = TabDébut To TabFin
        Egalités(j) = i
        Anc = i
        i = Ref(i)
        j = j + 1
        If Mémo(i) <> Mémo(Anc) Or n = TabFin Then
            If j > TabDébut + 1 Then Call QuickSort(Egalités(), TabDébut, j - 1)
            For l = TabDébut To j - 1
                Pos(Egalités(l)) = NbPassage
                NbPassage = NbPassage - 1
            Next l
            j = TabDébut
        End If
    Next n
    QuickRanking = Pos(): Exit Function
End If
' S'il faut retourner le tri sans doublons (et sans classement):
If ModeClassement = 4 Then
    NbPassage = TabDébut
    For n = TabDébut + 1 To TabFin
        If TabDonnées(n) <> TabDonnées(n - 1) Then NbPassage = NbPassage + 1
        TabDonnées(NbPassage) = TabDonnées(n)
    Next n
    ReDim Preserve TabDonnées(TabDébut To NbPassage)
    QuickRanking = Pos(): Exit Function
End If

QuickRanking = Pos()
End Function
```

```
'--------------------------------------------------------------------------------
Public Sub QuickSort(ByRef TabDonnées() As Variant, ByVal Gauche As Long, ByVal Droite As Long)
'--------------------------------------------------------------------------------
Dim i As Long, j As Long, Temp As Variant, Pivot As Variant

i = Gauche
j = Droite
Pivot = TabDonnées((Gauche + Droite) / 2)

Do
    While Pivot > TabDonnées(i): i = i + 1: Wend
    While TabDonnées(j) > Pivot: j = j - 1: Wend

    If j + 1 > i Then
        Temp = TabDonnées(i)
        TabDonnées(i) = TabDonnées(j)
        TabDonnées(j) = Temp
        j = j - 1: i = i + 1
    End If

Loop Until i > j

If Gauche < j Then Call QuickSort(TabDonnées(), Gauche, j)
If i < Droite Then Call QuickSort(TabDonnées(), i, Droite)

End Sub
```

You can optimize this algorithm for processing digital data by replacing the variable declarations "Variant" with "Long".

Annexe 4 : The QuickSort_AndRank algorithm (based on the Quicksort algorithm)

To return a ranking order.

```
'-------------------------------------------------------------------------------
Public Function QuickSort_AndRank(ByRef TabDonnées() As Variant, _
                Optional OrdreCroissant As Boolean = True, _
                Optional ModeClassement As Byte = 1, _
                Optional PivotFixe As Boolean = True) As Variant
'-------------------------------------------------------------------------------
Dim i As Long, Mini As Long, Maxi As Long
Dim Anc As Long, Classement As Long

Mini = LBound(TabDonnées)
Maxi = UBound(TabDonnées)
ReDim Ref(Mini To Maxi) As Long
Dim Pos() As Long

' Mémorise les données avant de les trier:
If ModeClassement < 3 Then
    ReDim Mémo(Mini To Maxi) As Variant
    For i = Mini To Maxi
        Ref(i) = i
        Mémo(i) = TabDonnées(i)
    Next i
Else
    For i = Mini To Maxi: Ref(i) = i: Next i
End If

' Trie les données:
If OrdreCroissant = True Then
    Call QS(TabDonnées(), Ref(), Mini, Maxi, PivotFixe)
Else
    Call QSDEC(TabDonnées(), Ref(), Mini, Maxi, PivotFixe)
End If

' S'il ne faut retourner que le classement:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If ModeClassement = 3 Then
    QuickSort_AndRank = Ref()
    Exit Function
End If

' Alimente TabDonnées du Tri:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~
If ModeClassement = 0 Then
    For i = Mini To Maxi
        TabDonnées(i) = Mémo(Ref(i))
    Next i
    QuickSort_AndRank = Pos()
    Exit Function
End If

' Alimente TabDonnées du Tri et retourne un classement où les égalités ont le même classement:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If ModeClassement = 1 Then
    ReDim Pos(Mini To Maxi) As Long
    Anc = Mini
    For i = Mini To Maxi
        TabDonnées(i) = Mémo(Ref(i))
        Classement = Classement + 1
        Pos(Ref(i)) = Classement
        If TabDonnées(i) = TabDonnées(Anc) Then Pos(Ref(i)) = Pos(Ref(Anc))
        Anc = i
    Next i
    QuickSort_AndRank = Pos()
    Exit Function
End If
```

```
' Initialise les variables pour la méthode 2:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Dim j As Long, l As Long, AncVal As Variant
ReDim Pos(Mini To Maxi) As Long

' Alimente TabDonnées dans l'ordre croissant du Tri et
' retourne un classement où les égalités distinguent l'ordre d'origine:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If OrdreCroissant = True Then

    ' Alimente TabDonnées du Tri et Retourne un classement où
    ' les égalités distinguent l'ordre d'origine:
    If ModeClassement = 2 Then

        j = Mini: AncVal = Mémo(Ref(j))
        ReDim Egalités(Mini To Maxi) As Variant  ' Tableau des égalités.

        For i = Mini To Maxi
            TabDonnées(i) = Mémo(Ref(i))
            If TabDonnées(i) = AncVal Then ' Si nouvelle égalité alors:
                Egalités(j) = Ref(i) ' Ajoute la donnée dans le tableau des égalités.
                j = j + 1            ' Compte le nombre d'égalités.
            Else ' Si plus d'égalité:
                If j > Mini + 1 Then Call QuickSort(Egalités(), Mini, j - 1) ' Si plusieurs égalité
                For l = Mini To j - 1 ' Boucle sur les égalités:
                    Classement = Classement + 1  ' Incrémente le classement.
                    Pos(Egalités(l)) = Classement ' Indique le classement.
                Next l
                ' Place cette donnée dans le tableau des égalités:
                j = Mini
                Egalités(j) = Ref(i)
                j = j + 1
            End If
            AncVal = Mémo(Ref(i)) ' Nouvelle référence de comparaison.
        Next i

        ' Vide le tableau des égalités:
        If j > Mini + 1 Then Call QuickSort(Egalités(), Mini, j - 1) ' Si plusieurs égalités.
        For l = Mini To j - 1 ' Boucle sur les égalités:
            Classement = Classement + 1  ' Incrémente le classement.
            Pos(Egalités(l)) = Classement ' Indique le classement.
        Next l
    End If

    QuickSort_AndRank = Pos()
    Exit Function
End If
```

```
' Alimente TabDonnées dans l'ordre Décroissant du Tri et
' retourne un classement où les égalités distinguent l'ordre d'origine:
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
If ModeClassement = 2 Then

    j = Mini: AncVal = Mémo(Ref(j))
    ReDim Egalités(Mini To Maxi) As Variant  ' Tableau des égalités.

    For i = Mini To Maxi
        TabDonnées(i) = Mémo(Ref(i))

        If TabDonnées(i) = AncVal Then ' Si nouvelle égalité alors:
            Egalités(j) = Ref(i) ' Ajoute la donnée dans le tableau des égalités.
            j = j + 1            ' Compte le nombre d'égalités.
        Else ' Si plus d'égalité:
            If j > Mini + 1 Then Call QuickSortDESC(Egalités(), Mini, j - 1) ' Si plusieurs égalités
            For l = Mini To j - 1 ' Boucle sur les égalités:
                Classement = Classement + 1  ' Incrémente le classement.
                Pos(Egalités(l)) = Classement ' Indique le classement.
            Next l
            ' Place cette donnée dans le tableau des égalités:
            j = Mini
            Egalités(j) = Ref(i)
            j = j + 1
        End If
        AncVal = Mémo(Ref(i)) ' Nouvelle référence de comparaison.
    Next i

    ' Vide le tableau des égalités:
    If j > Mini + 1 Then Call QuickSortDESC(Egalités(), Mini, j - 1) ' Si plusieurs égalités.
    For l = Mini To j - 1 ' Boucle sur les égalités:
        Classement = Classement + 1  ' Incrémente le classement.
        Pos(Egalités(l)) = Classement ' Indique le classement.
    Next l
End If

QuickSort_AndRank = Pos()
End Function
```

```
'-------------------------------------------------------------------------------
Private Sub QS(ByRef TabDonnées() As Variant, ByRef Ref() As Long, _
               ByVal Gauche As Long, ByVal Droite As Long, ByVal PivotFixe As Boolean)
'-------------------------------------------------------------------------------
Dim i As Long, j As Long, Temp As Long, ValQS As Variant

i = Gauche
j = Droite
If PivotFixe = True Then
    ValQS = TabDonnées(Ref((Gauche + Droite) / 2))
Else
    ValQS = TabDonnées(Ref(Gauche + (Rnd() * (Droite - Gauche))))
End If

Do
    While ValQS > TabDonnées(Ref(i)): i = i + 1: Wend
    While ValQS < TabDonnées(Ref(j)): j = j - 1: Wend

    If j + 1 > i Then
        Temp = Ref(i)
        Ref(i) = Ref(j)
        Ref(j) = Temp
        j = j - 1: i = i + 1
    End If

Loop Until i > j

If Gauche < j Then Call QS(TabDonnées(), Ref(), Gauche, j, PivotFixe)
If i < Droite Then Call QS(TabDonnées(), Ref(), i, Droite, PivotFixe)

End Sub

'-------------------------------------------------------------------------------
Private Sub QSDEC(ByRef TabDonnées() As Variant, ByRef Ref() As Long, _
                  ByVal Gauche As Long, ByVal Droite As Long, ByVal PivotFixe As Boolean)
'-------------------------------------------------------------------------------
Dim i As Long, j As Long, Temp As Long, ValQS As Variant

i = Gauche
j = Droite

If PivotFixe = True Then
    ValQS = TabDonnées(Ref((Gauche + Droite) / 2))
Else
    ValQS = TabDonnées(Ref(Gauche + (Rnd() * (Droite - Gauche))))
End If

Do
    While ValQS < TabDonnées(Ref(i)): i = i + 1: Wend
    While ValQS > TabDonnées(Ref(j)): j = j - 1: Wend

    If j + 1 > i Then
        Temp = Ref(i)
        Ref(i) = Ref(j)
        Ref(j) = Temp
        j = j - 1: i = i + 1
    End If

Loop Until i > j

If Gauche < j Then Call QSDEC(TabDonnées(), Ref(), Gauche, j, PivotFixe)
If i < Droite Then Call QSDEC(TabDonnées(), Ref(), i, Droite, PivotFixe)

End Sub
```

Annexe 5 : Comparison of the three algorithms.

	QuickSort			QuickSort_AndRank				QuickRanking							
	Sorting Without Ranking			Sorting Without Ranking	Sorting and ranking Method 1	Sorting and ranking Method 2	Ranked only Method 3	Sorting Without Ranking		Sorting and ranking Method 1		Sorting and ranking Method 2		Ranked only Méthod 3	
100,000 integers :		(A_1)	(A_2)						(B_0)		(B_1)		(B_2)		(B_3)
Totally random distribution	0,18	-7%	0%	0,19	0,20	0,21	0,18	0,18	-7%	0,19	-7%	0,20	-7%	0,17	-7%
20% first sorted and then randomized	0,18	-7%	18%	0,19	0,20	0,21	0,18	0,15	-21%	0,16	-20%	0,17	-19%	0,14	-22%
Randomized sorted except last 20%	0,18	-6%	0%	0,19	0,20	0,21	0,18	0,18	-6%	0,19	-7%	0,20	-6%	0,17	-8%
500,000 integers :															
Randomized, values between 1 and 500,000	0,98	-19%	-24%	1,20	1,26	1,38	1,12	1,29	7%	1,41	11%	1,52	10%	1,20	7%
Randomized, values between 1 and 100	0,93	-20%	-3%	1,17	1,26	1,68	1,09	0,96	-18%	1,08	-14%	1,35	-20%	0,88	-19%
100,000 alphanumeric between 1 and 100 chars :															
Totally random distribution	0,92	8%	28%	0,85	0,88	0,90	0,75	0,72	-16%	0,77	-13%	0,77	-15%	0,61	-18%
20% first sorted and then randomized	0,92	9%	47%	0,84	0,88	0,90	0,75	0,63	-25%	0,67	-23%	0,67	-25%	0,53	-29%
Randomized sorted except last 20%	0,90	10%	29%	0,82	0,85	0,88	0,73	0,70	-15%	0,75	-12%	0,75	-15%	0,60	-17%
Randomized sorted :															
List of 36,680 common France	0,29	8%	30%	0,27	0,28	0,30	0,25	0,22	-16%	0,23	-18%	0,23	-21%	0,20	-20%
100,000 alphanumeric of 20 chars	0,80	0%	22%	0,80	0,84	0,86	0,71	0,66	-18%	0,71	-16%	0,72	-16%	0,59	-16%
500,000 alphanumeric of 20 chars	4,35	-8%	0%	4,70	4,88	4,97	4,44	4,31	-8%	4,61	-6%	4,68	-6%	3,96	-11%
100,000 alphanumeric of 1 to 250 chars	1,08	12%	39%	0,96	1,00	1,03	0,83	0,78	-19%	0,83	-17%	0,84	-19%	0,64	-23%
500,000 alphanumeric of 1 to 250 chars	6,32	11%	30%	5,70	5,82	5,99	4,98	4,88	-14%	5,19	-11%	5,25	-12%	4,16	-16%
1,000,000 alphanumeric of 1 to 250 chars	13,59	8%	26%	12,53	12,81	13,11	11,18	10,75	-14%	11,44	-11%	11,53	-12%	9,28	-17%
100,000 alphanumeric of 1,000 chars	1,92	13%	71%	1,70	1,74	1,78	1,32	1,12	-34%	1,17	-33%	1,18	-34%	0,72	-45%

Method 1 : the data equal have the same order

Method 2 : order data equal respect the original order.

Method : does not return sorting data but only the ranking without management equalities.

(A1) : Comparison of ranking without sorting between Quicksort and QuickSort_AndRank.

(A2) : Comparison of ranking without sorting between Quicksort and QuickRanking

(B) : Comparison between QuickRanking and QuickSort_AndRank : B0 = Sorting without ranking, B1 = Sorting and ranking method 1, B2 = Sorting and ranking method 2, B3 = ranking only method 3.

Quick Ranking does not use the additional analysis option. This option is not useful on random data.

Annex 6: analysis of a sample of the data in the list allows you to determine whether or not to enable the option of additional tests.

QuickRanking offers optional complementary analyses to accelerate the processing, based on the last analysised item - research of series and duplicates for immediate processing, research of series non-contiguous for a rapid update of the table of already classified items.
These analyses are performing with partially classified lists or if there are many equalities, but are counter-productive with random lists.
That is why these analyses are optional in QuickRanking in order not the run on lists of data that you know they are random, and thus save processing time.

The dilemma thus arises when you do not know in advance if the list of data to sort will be or not random lists. Because in some cases activating the option will speed up incredibly the processing or, on the contrary will slow it down.
The solution I suggest, is to take at random a sample of data from the list, and analyse it with the selected option. QuickRanking will feed in his argument taken by reference, *TauxTest*, the rate of successful tests. If this number represents a significant percentage, in the *TauxConformite argument* of the function, it can be estimated that it must activate the option.

The larger the sample size, the more the test will be representative, but the longer time it will take. Thus, I have limited the size of the sample to 3% of the size of the original list. Conversely if the sample is less than 20 items, it is considered non-significant and the function reverses True without making a test. By default the size of the sample is 1/1000 of the list.

```
'---------------------------------------------------------------------------
Private Function TesterNiveauQR(ByRef MonTableau() As Variant, _
                        ByVal TauxConformité As Long, _
                        Optional ByVal PcEchantillon As Double = 0.1) As Boolean
'---------------------------------------------------------------------------
Dim Début As Long, Fin As Long, TailleEchantillon As Long, i As Long, l As Long
Début = LBound(MonTableau())
Fin = UBound(MonTableau())

' Initialisation des variables:
TailleEchantillon = (Fin - Début) * PcEchantillon / 100

' Controle la taille de l'échantillon pris au hasard dans la liste:
If TailleEchantillon > Fin * 0.03 Then TailleEchantillon = Fin * 0.03
If TailleEchantillon < 20 Then TesterNiveauQR = True: Exit Function
ReDim MonTest(Début To TailleEchantillon) As Variant
Do
    i = Rnd() * Fin
Loop While Début + i + TailleEchantillon > Fin

For l = i To i + TailleEchantillon
    MonTest(Début) = MonTableau(l): Début = Début + 1
Next l

' Compte le nombre de tests fructueux avec l'option à Vrai:
i = 0: Call QuickRanking(MonTest(), True, 3, True, i)

' Retourne Vrai si les tests sont efficaces dans au moins TauxConformité% des cas:
If i > TauxConformité Then TesterNiveauQR = True

End Function
'---------------------------------------------------------------------------
```

Bibliography:

QuickSort: Hoare, C. A. R. « Partition: Algorithm 63, » « Quicksort: Algorithm 64, » and « Find: Algorithm 65. » Comm. ACM 4, 321-322, 1961.

QuickSort_AndRank (based on the Quicksort algorithm): Laurent Ott

QuickRanking: Laurent Ott